Amendments to the Specification:

Please replace the paragraph beginning at page 5, line 3, with the following rewritten paragraph:

Referring now to the drawings, and more particularly to Figure 1, there is shown a block diagram of the adaptive integration activity management framework for on demand business process integration according to the invention. The architecture is roughly divided into three parts: 1) the input receiving and processing components, comprising the portal/requester 11 that receives incoming messages, and the action manager 12 that parses and processes the messages, 2) the adaption layer 13 that provides the interface between the action manager 12 and existing components, 3) and the existing components, which may include legacy applications such as SCM (Supply Chain Management) 14 and ERP (Enterprise Resource Planning) 15 and legacy application connectors/adaptors 16, FTP (File Transfer Protocol) 17, and RFTP (Reliable File Transfer Protocol) 18, and the like. The idea is to integrate the three parts without having to change the first part (ActionManager) and the third part (existing components) by supplying only the implementation of the adaptation layer (the second part) for a specific integration activity of an application to be integrated.

Please replace the paragraph beginning at page 5, line 19, with the following rewritten paragraph:

The action manager 12, which is the engine of the framework, receives collaborative messages (or CxP "Collaborative Exchange Protocol" messages) ~~messages~~ from a design partner side, which can be a Web portal. ~~In each~~ Each message~~, it~~ contains meta data or annotations describing the documents to be

exchanged, such as the file name, size, author, application to use to open the design file, etc. In addition, annotations can also specify integration activities to be performed, representing new application to be integrated, such as FTP, reliable file transfer (RFT) or an invocation to an legacy adaptor. Also, an alternative data source to the Action Manager, in addition to collaborative messages, is an RDF string. The adaptation layer 13 is an integration layer that provides isolation from the action manager 12 to the applications to be integrated. There is a pre-defined adaptation layer interface that the action manager 12 knows about and can invoke the methods of the adaptation layer when an implementation of the interface is in place. When there is a new integration activity for an application, ~~there~~ this requires an implementation of an adaptation layer, or an adaptor, for that application using the pre-defined adaptation layer interface. The newly implemented adaptation layer 13 will be able to get "plugged-in" to the action manager, which can automatically invoke the standard method defined in the interface for the adaptor, which in turn invokes the new application. Notice that neither the action manager nor the application needs to be modified for this integration to take place.

Please replace the paragraph beginning at page 6, line 14, with the following rewritten paragraph:

There is an "on-demand" aspect in ~~what~~ the integration activity, i.e., the activity can be specified dynamically via the meta data or annotation at runtime. The value of the activity can be any of a range of integration activities that the receiving partners can support, i.e., having an adaptation layer built and deployed. That is, when a sender is creating a collaborative message to send to the receiving partner, the sender can "optionally" specify what type of integration action is needed as part of the collaborative message. The integration action does not need to be pre-defined in a schema to be sent. In addition, a different instance of the same type of collaborative

message can have different actions as well. This is because a collaborative message accounts for any number of annotations that communicating parties can understand, and an action is just one of the understood annotations. Thus, the integration can be optionally specified by the sender, and understood and processed by the receiver. On the other hand, if a sender were to send an RosettaNet or EDI (Electronic Data Interchange) message, there must be a message that already ~~specify~~ specifies the integration action as part of the message to be understood and processed by the receiver, which is not flexible. Therefore, EDI or RosettaNet do not provide the agility and flexibility required by collaborative business processes for their limitation to the rigid data models. Furthermore, the design collaboration process is creative and dynamic; it is not possible to pre-define all actions ahead of time. The approach taken by this invention gives the design partner or trading partner a more flexible way to trigger a new or different integration activity at any on need basis. The sender is not restricted to only a certain type of collaborative ~~messages~~ message that can trigger a certain type of integration ~~activities~~ activity, thus enabling the on-demand capability.

Please replace the paragraph beginning at page 7, line 11, with the following rewritten paragraph:

The ActivityChain ontology 45 that guides the action manager 12 provides flexibility and granularity for an adaptive and on-demand mechanism to integrate new applications to an existing environment. The ActivityChain ontology is based on Resource Description Framework (RDF), which is a W3C Recommendation for describing resources and defined in DAML+OIL language. The ontology provides the schema for runtime RDF annotation exchange.

Please replace the paragraph beginning at page 8, line 2, with the following rewritten paragraph:

Ontology provides the following purposes:

1. To share common understanding of the structure of information among people or software agents
2. To enable reuse of domain knowledge
3. To make domain assumptions explicit
4. To separate domain knowledge from the operational knowledge
5. To analyze domain knowledge

And we believe high-level business process integration knowledge can be separated from the "operational" hard code and ~~capture~~ captured in terms of ontology~~. Thus,~~ and thus be reused among different software components. Further, a business process integration condition is captured explicitly by ontology, which makes the monitoring and self-managing possible. Our ActivityChain ontology is an instance of ontology in the business process integration domain.

Please replace the paragraph beginning at page 8, line 27, with the following rewritten paragraph:

Figure 2 shows how ActivityChain ontology is used to capture the activities-based business process flow in a design process integration scenario. This is a diagram showing a scenario that ActivityChain ontology can be used in design of a collaborative process. Wave 1, Wave 2, Wave 3 represents ~~a~~ an outline procedure of a new product design, e.g., a new notebook computer model design. The overall design follows a top-down approach. As time goes from Wave 1 to Wave 2, to Wave 3, the broad design goals become more clear and more detailed. But the timeline of Wave 1, Wave 2 and Wave 3 are not strictly ~~seperated, sometimes~~ separated. Sometimes a design change in Wave 3 may bring back and incur changes in Wave 2, even Wave ~~3~~ 1.

Please replace the paragraph beginning at page 9, line 9, with the following rewritten paragraph:

Wave 1 is the high-level design stage. Starting from the left block, to design a new notebook computer model, the first step is to collect the design requirements from a broad range of customers. We called this step Requirement Capture. After that, a design team will be formed. Next step is to start Product design. But at this time, the product ~~design~~ designs are limited to the high-level goals for new notebook computer model, like the performace metric goals, appearance, etc. After all the designs are completed, they are sent to be manufactured.

Please replace the paragraph beginning at page 9, line 25, with the following rewritten paragraph:

We believe that our ActivityOntology is ~~such a~~ a very good tool to ~~be able to~~ capture the dynamic behaviors in the design collaborative process above. We show an example of this usage as follows: For example, in Wave 2 the CPU design can be decomposed into Chip1 design, Chip 2 design, Chip 3 design, etc. At this level, the ~~design~~ designer may only know the set of outsourcing candidates for doing Chip 1 design and could not decide which one is picked yet. So the CPU design process in Wave 2 only specified that the most reliable and most reasonable price design partner will be picked for Chip 1 design. Coming down to Wave 3, all the Chip 1 design candidates will be contacted and the best Chip 1 design partner will be selected. After the selecting process, the CPU design process is detailed to which design partner will do the Chip 1 design in Wave 3.

Please replace the paragraph beginning at page 10, line 24, with the following rewritten paragraph:

The logical structure of ActivityChain ontology is shown in Figure 3. The top-level entity is Class Activity. It has a DataTypeProperty securityHandler and an ObjectProperty actname. The ObjectProperty actname has a range which is Class Actname. And Actname is a collection which enumerates GridFTP,FTP, HTTP, Inv-service, Inv-Appl and ~~Scarch-Annt~~ Search-Annot.

Please replace the paragraph beginning at page 11, line 4, with the following rewritten paragraph:

In DAML+OIL syntax, subClasses are specified as a collection that inherits the superClass with some restriction on certain properties. Thus, subClass GridFTP is a collection that inherits Class Activity with two DataTypeProperty restrictions. One is source and the other is destination. These properties have to be unique for this class. Similarly subClass FTP is a collection that inherits Class Activity with two DataTypeProperty restrictions. One is getFrom and the other is sendTo. They have to be unique, too.

Please replace the paragraph beginning at page 15, line 14, with the following rewritten paragraph:

And the flexible linkage, which provides ~~constrains~~ constraints to the relationship between the individual activities, is defined in Table 2. For ~~simplicity~~ the purpose of simplicity, we choose to use SRML (Simple Rule Markup Language) proposed by Margaret Thorpe and Charighai Ke, ILOG, S.A. as our expression language to explain our ideas. Other rule languages like BRML (Business Rules Markup Language) from IBM can be used ~~as~~ in the same manner.

Please replace the paragraph beginning at page 17, line 12, with the following rewritten paragraph:

Not only a simple linkage such as below

If File To TransferfileSize >=100MB, switch to GridFTP.
If File To TransferfilleSize < 100MB, switch to FTR

can be expressed in our ActivityChain Ontology, but also more complex ~~relationship~~ relationships between individual activities can be expressed freely and captured by the Action ~~Managerwhich~~ Manager, which helps smartly monitoring and managing the activities. For example, an exception like file transferring failure in GridFTP, a restart action can be invoked after a certain amount of time.

Please replace the paragraph beginning at page 18, line 1, with the following rewritten paragraph:

The action manager components ~~is~~ are shown in Figure 4. The key components in action manager are the activity parser 41, the event capture 42, the controller 43, the access control utility 44, the activity ontology 45, the exception handler 46, and the adaptation layers 47_1 to 47_n. The purpose of the activity parser 41 is to retrieve the integration activity name from an input parameter, such as an RDF String or an annotated CxP message. The event capture 42 captures the event, such as a designer having finished a CAD (computer aided design) design document. This act may later trigger an activity, which is to notify the design partner that the design document is done. The access control utility 44 checks the authorization of all access to the resources/application integrated and managed by the activity manager, ensuring all ~~access~~ accesses are authorized.

Please replace the paragraph beginning at page 18, line 24, with the following rewritten paragraph:

Exception handler 46 monitors all activities at runtime. When an exception occurs, it makes decisions based on the specification in the Activity ontology and triggers the appropriate action of the Controller. Adaptation layers 47_1 to 47_n are another key component, which realizes the "adaptive" feature of the action manager. Each adaptation layer has to be created for each integration activity. All the adaptation layers have the same input interface which will accept a XML (eXtensible Markup Language) string as its input parameter. One particular adaptation layer will get the parameters from the XML string and create the required parameter invocation sequence, then do actual invocation on the corresponding application. Each adaptation ~~lay~~ layer itself is web-service based, therefore, platform-independent. To construct a collaborative message or RDF String correctly, ActivityChain ontology has to be consulted.